Testing rotation sensor drivers with LEGO robots and other adventures in the Linux IIO subsystem

Speaker: David Lechner

Dossier

- David Lechner
- Embedded software engineer, BayLibre
  - embedded software consultancy based in Nice, France, with ~60 engineers around the world contributing to open source projects like Linux, U-Boot, Android, Zephyr and the GNU toolchain.
- LEGO MINDSTORMS robotics expert
  - ev3dev - Debian Linux on LEGO MINDSTORMS EV3
  - Pybricks - MicroPython on LEGO Powered Up smart hubs

Choose your own adventure

- The adventure of the remote IIO developers
- The adventure of the resolver to digital converter
- The adventure of the SPI offload
- The adventure of the IIO backend framework
- The adventure of the IIO DMABUF

The adventure of the remote IIO developers

Our lab setup

Our lab setup

Anywhere

Lab

Mini PC

Serial console

Power Adapter

Power Switch

ZedBoard

ADC Eval Board

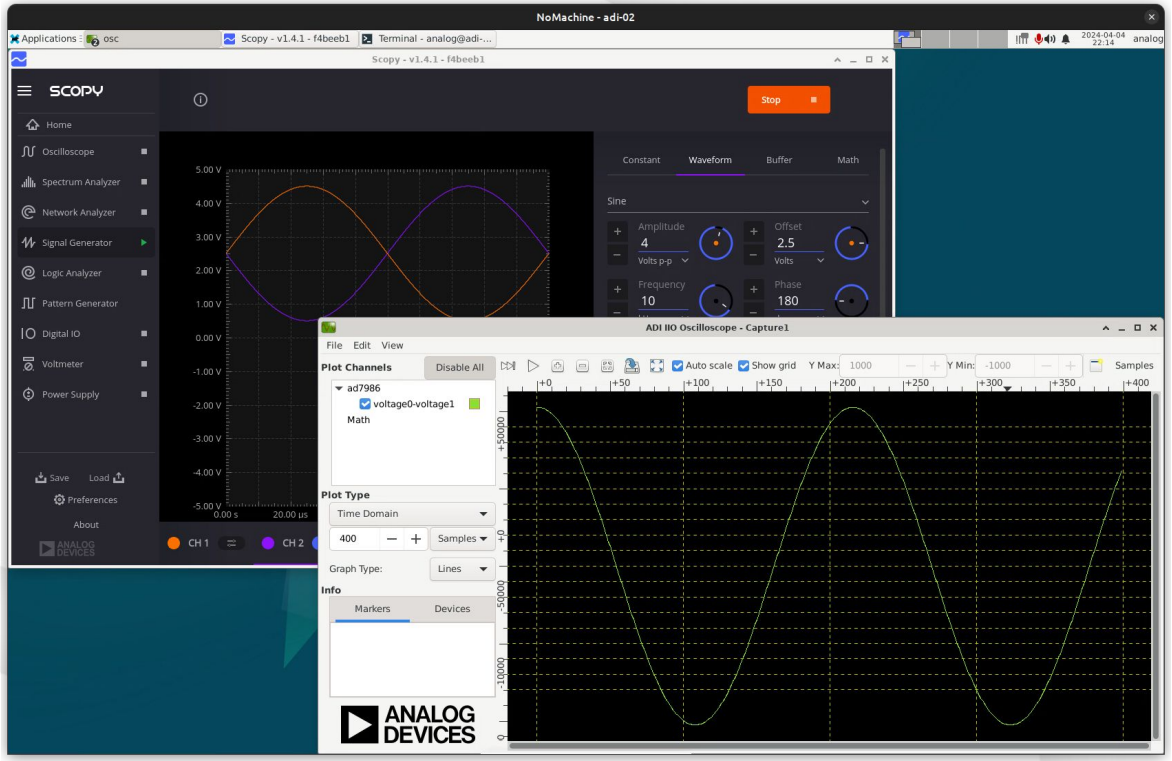
Signal Generator

Logic Analyzer

Oscilloscope

VPN

Apps running on Mini PC



▸ The adventure of the remote IIO developers

What could possibly go wrong (with remote hardware)?

Biggest pain points:

- Sharing resources between devs (mini-pc, logic analyzer, etc.)
- Updating boot files / recovering from bad kernel, etc.
- Setting up / swapping boards in the lab
- Debugging physical electronics issues

How to remotely update boot files on your development board?

ZedBoard: Xilinx Zynq 32-bit ARM SoC/FPGA, Kuiper Linux (Raspberry Pi OS derivative), FMC connector

Boots from microSD card with **BOOT.BIN** that contains U-Boot and FPGA bitstream, Linux kernel **uImage** and **devicetree.dtb**.

- We tried SD card mux.
  - It didn't work due to electrical issues.
- We tried TFTP boot and NFS file system.
  - It was complex to set up and made userspace really slow.
  - And it doesn't handle the BOOT.BIN file.

How to remotely update boot files on your development board?

- We tried keeping it simple.
  - Boot files are written to the SD card via SSH on a running dev board.

    ```
    scp uImage devicetree.dtb root@zed-1:/boot/
    ```

  - If we load a bad kernel, we can recover remotely using the U-Boot shell.

    ```
    setenv kernel_image backup/uImage
    boot
    ```

  - Updating BOOT.BIN is a bit more risky.
    - But… we only managed to mess that up less than once per month

Is it possible to do remote development with IIO-type hardware?

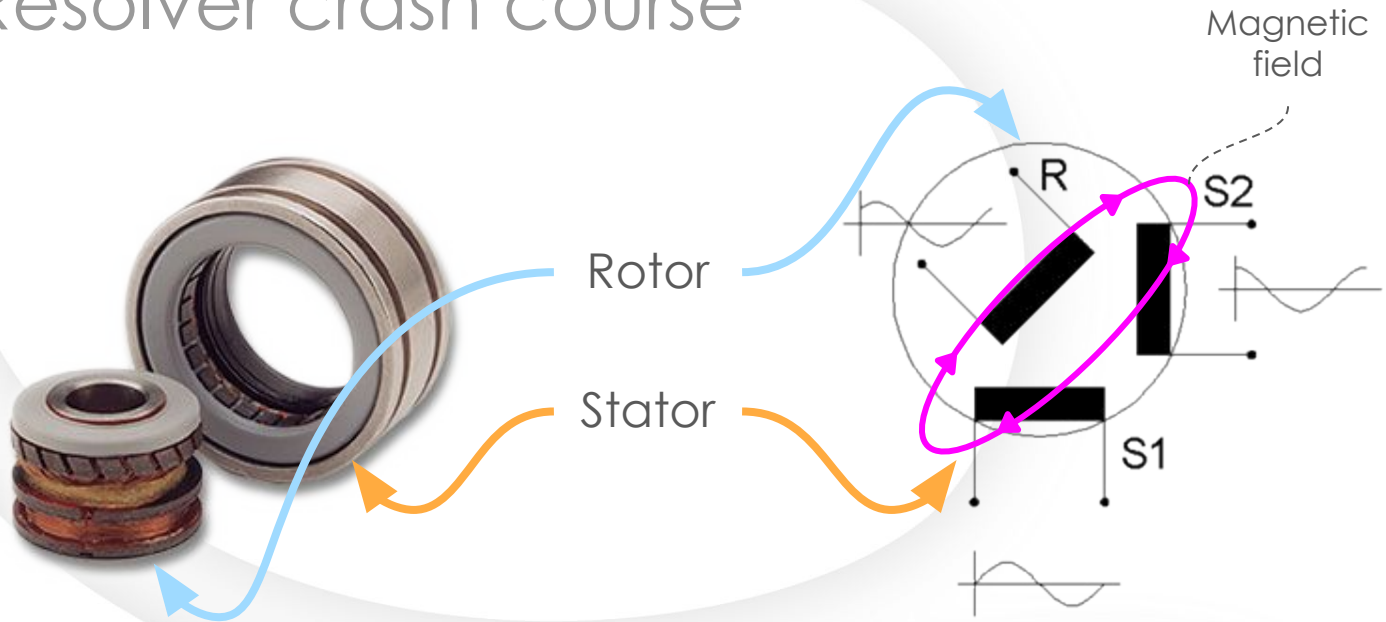
- Why not have the hardware and the engineer in the same place?
  - Some hardware needs to be shared by multiple remote engineers
  - Shipping costs add up and items get lost in the mail
  - Not all remote engineers are experts at soldering or have all required test equipment
- Exceptions
  - When only one engineer needs the hardware and has the tools/experience
  - When there is a duplicate board
  - For initial bring-up of particularly complex hardware

Overall, we've been making it work.

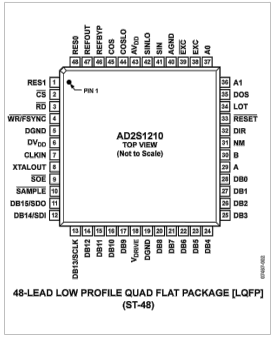
Our first Linux driver using this setup looked like this…

The adventure of the resolver to digital converter

Resolver crash course



Resolver



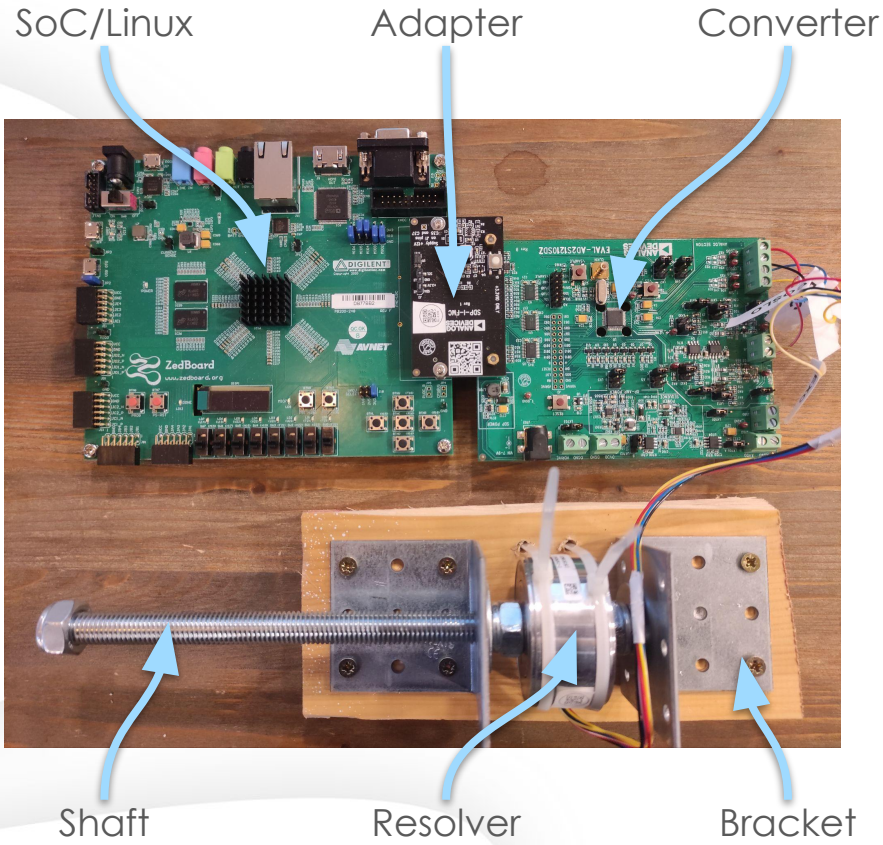
Converter

Outputs angle, speed and faults

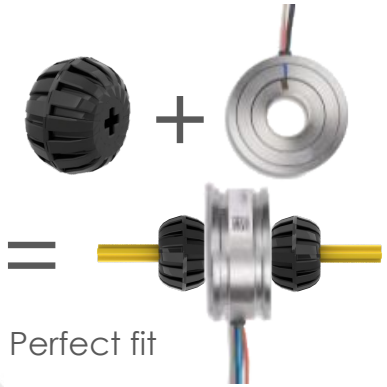
Test setup V1

Issues:

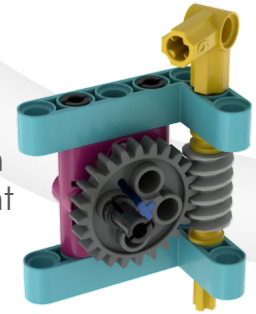
- Human powered
  - Not available 24/7
  - No constant speed setting
  - No high speed setting
- Alignment
  - Rotor and stator must not touch
  - 0.15 mm air gap



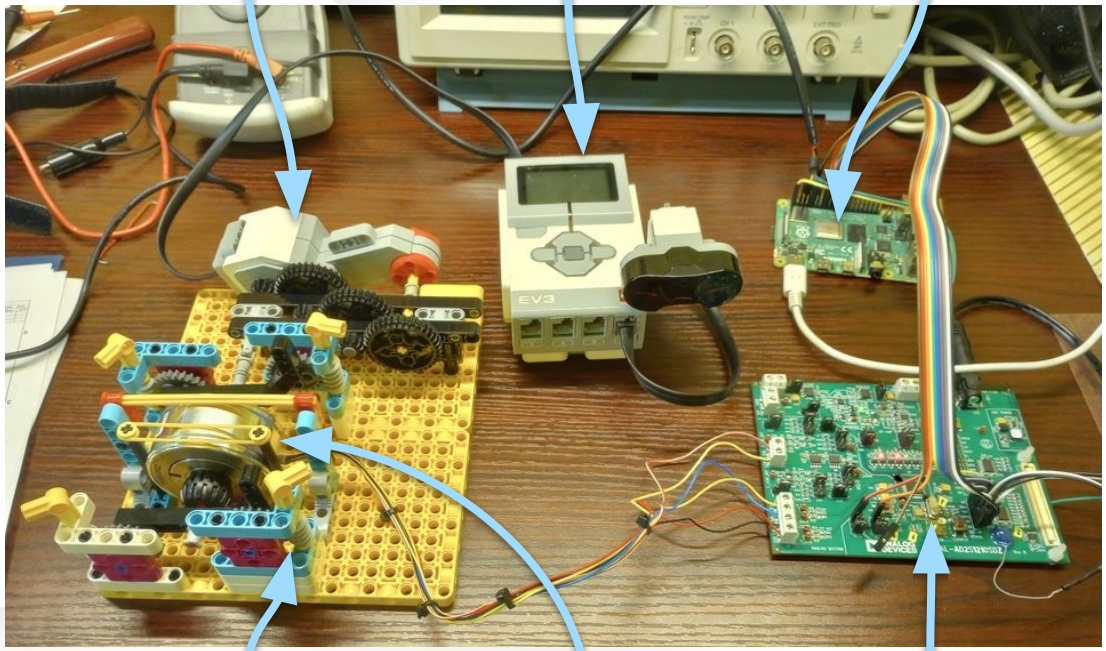
Test setup V2



Perfect fit



Precision alignment



Demo video

How to get a driver out of staging?

There was already an AD2S1210 resolver driver in staging… for 13 years!

- Fix the bugs first
  - Always best practice
- Then fix the userspace interface
  - Try to use standard IIO attributes
- And write the devicetree bindings
  - Watch out for undocumented properties already in the driver!
- Now you can move it out of staging
  - `git format-patch --no-renames`
- Finally add new features
  - Example: implementing previously unsupported hardware feature

Upstream support: mission accomplished.

It just took some ingenuity … and 40 patches!

Speaking of upstream…

The adventure of the SPI offload

SPI offloading

Problem:

- We need to get **millions** of samples **per second** from an ADC via SPI
- Typical embedded Linux can get **10k** to **100k** samples per second
- And there are multiple sources of **jitter**

What's wrong with this picture?

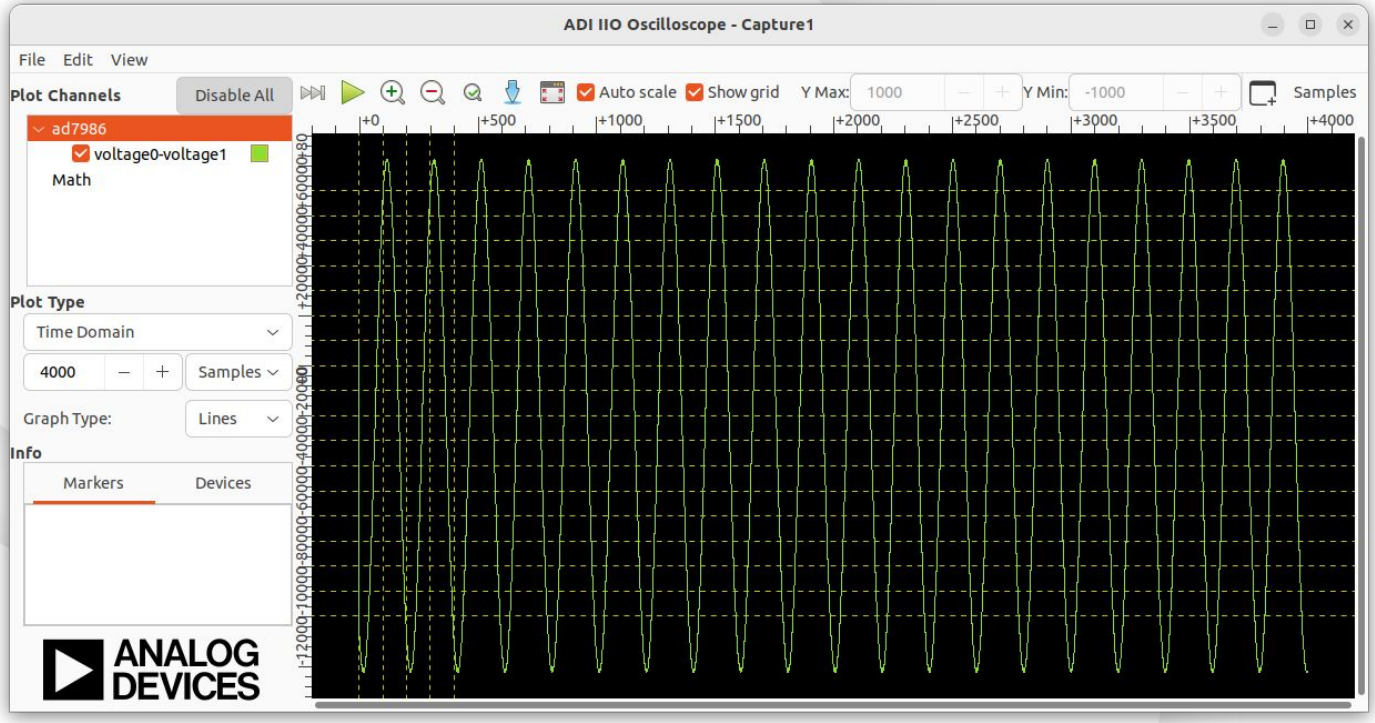
Huge gap in data due to missed hrtimer trigger interrupts

SPI offloading

Solution:

- SPI transfers done without CPU intervention
  - “Record” and play back SPI transfer
  - Hardware trigger (e.g. PWM)
  - RX data streamed to DMA buffer
- Virtually eliminates jitter
- Allows *way* more samples per second

2 million samples per second over SPI w/ offload



10kHz reference signal

- The adventure of the SPI offload

Road to mainline for SPI offload

- [PATCH 00/13] spi: axi-spi-engine: add offload support
  - Quite a few criticisms (too complex, not general enough, ...)
  - Suggested breaking it down into smaller parts
- [PATCH v2 0/5] spi: add support for pre-cooking messages
  - Optimization for any SPI peripheral driver that repeats xfers
  - No special hardware required to use this
  - Landed in kernel v6.9
- Next step?

> [GIT PULL] spi: Updates for v6.9
>
> This release sees some exciting changes from David Lechner which implements some optimisations that have been talked about for a long time which allows client drivers to pre-prepare SPI messages for repeated or low latency use. This lets us move work out of latency sensitive paths and avoid repeating work for frequently performed operations. As well as being useful in itself this will also be used in future to allow controllers to directly trigger SPI operations (eg, from interrupts).

Want to help?

Ideas for other applications that could benefit from improved SPI performance:

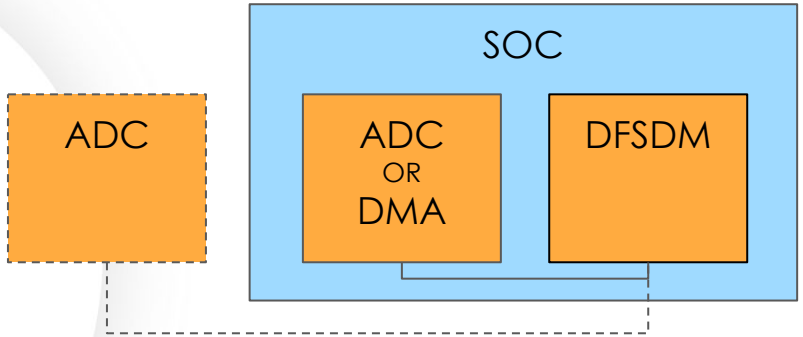
- Use it in your favorite IIO driver
- Move DMA mapping to `spi_optimize_message()`
- Improve performance of SPI-based CAN controllers
- Improve performance of `spi-mem` by making use of lookup tables in Freescale QuadSPI or NXP FlexSPI controllers
- Write a SPI offload controller driver + firmware for a microcontroller embedded in a SoC

The adventure of the IIO backend framework

Composite IIO devices

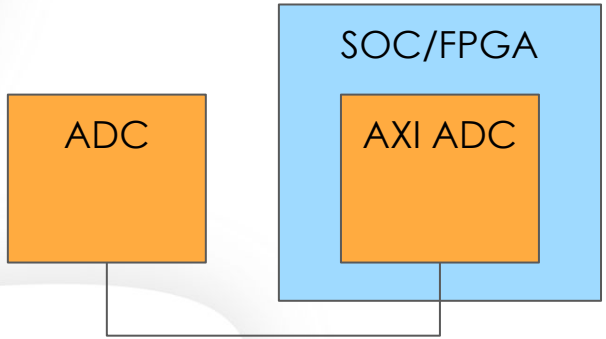
STM32 needed a driver for DFSDM IP block

Digital Filter for Sigma-Delta Modulator



ADI had a driver for AXI ADC IP block but needed to make it more generic

FPGA IP block for high speed data acquisition



IIO backend framework

Frontend

Device with primary input or output

Example: ADC or DAC

Backend

Provides extra function

Example: Digital filter

[PATCH v11 0/7] iio: add new backend framework

From: Nuno Sa via B4 Relay <devnull+nuno.sa.analog.com@kernel.org>

(landed in kernel v6.9)

Using the IIO backend framework

Frontend:

- Devicetree
  - **`iio-backends <&backend>;`**
- Driver:
  - Registers/owns **`iio:deviceX`**
  - **`devm_iio_backend_get()`**
  - Calls **`iio_backend_ops`**

Backend:

- Devicetree
  - **`#iio-backend-cells = <0>;`**
- Driver
  - **`devm_iio_backend_register()`**
  - Implements **`iio_backend_ops`**

The adventure of the IIO DMABUF

IIO DMABUF support

Problem:

We need to get **64 MiB** of sample data from **IIO** to **USB** ASAP

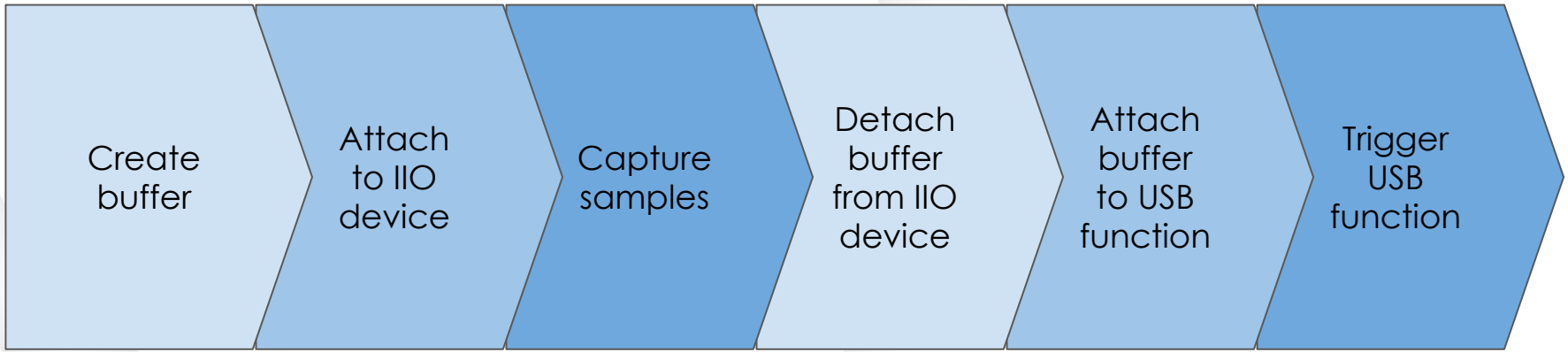
Solution:

[PATCH v9 0/6] iio: new DMABUF based API

From: Paul Cercueil <paul@crapouillou.net>

(should be landing in 6.10 kernel)

How IIO DMABUF works




```
> This drastically increases the throughput, to about 274 MiB/s over a
> USB3 link, vs. 127 MiB/s using IIO's fileio interface + write() to the
> FunctionFS endpoints, for a lower CPU usage (0.85 vs. 0.65 load avg.).
```

The end